# Simulation of Wireless Multi-* Networks in NS-2

Laurent Paquereau
laurent.paquereau@q2s.ntnu.no

Bjarne E. Helvik
bjarne@q2s.ntnu.no

Centre for Quantifiable Quality of Service in Communication Systems*
Norwegian University of Science and Technology, Trondheim, Norway

## ABSTRACT

Multi-technology, multi-homed, multi-hop, multi-interface, multi-channel, multi-route, multi-destination... Emerging wireless networks are *multi-** networks. Emerging wireless networks are also no longer stand-alone and self-contained networks but connected to external networks. Simulating such complex systems requires advanced network simulation tools. The network simulator 2 (ns-2) is one of the most widely used simulators and has constantly been enriched to design, test and evaluate new network architectures and protocols. This paper motivates and presents the design and implementation of a network layer architecture that extends the functionality of ns-2 to support the aforementioned features. In particular, it provides support for multiple interfaces, potentially of different types, and multiple routing and forwarding protocols running on the same node.

## Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

## General Terms

Design

## Keywords

Ns-2, multi-* wireless networks, network layer architecture

## 1. INTRODUCTION

During the last decade, the foreseen potential of wireless multi-hop networks, combined with the rapid development of enabling technologies, have made them an active research area. In particular, an early focus was on protocols for standalone wireless mobile ad-hoc networks (MANET). This research has however been limited by the lack of well-defined targeted applications other than niche applications, typically battlefield operations. More recently, more specific and better defined applications such as wireless sensor networks (WSN), vehicular ad-hoc networks (VANET) and wireless mesh networks (WMN) have gained increasing attention. These networks are *multi-** networks. They are multi-hop, -radio, -channel, -destination, etc. Although they obviously have a lot in common with traditional MANET, e.g. multi-hop communication and self-configuration property, they distinguish themselves notably in that they should interwork with some external networks. General purpose protocols proposed for wireless ad-hoc networks are often not suited for any of these applications. They need to be adapted, or new protocols designed, taking into account the characteristics and requirements of these applications. Example of such characteristics are scale, node density, typical number of radio interfaces per node, traffic types and volumes, mobility, available computational capacity on each node, reliability of each node. Example of requirements include provisioning QoS guarantees or power-efficiency, . A number of research issues remain open at all layers of the protocol stack for each of these applications [6, 8, 17], and tools are needed to enable the design and the comparison of solutions. Considering the complexity of such systems, simulation is often used and the network simulator 2 (ns-2) [3] is one of the most widely applied tools by networking researchers.

The development of ns-2 started in the 1990's and, since then, it has constantly been extended to novel research areas, networking technologies and scenarios. In the past few years, several initiatives have aimed at enhancing, extending or redesigning the wireless network modelling in ns-2 and improving the simulation accuracy. Most of these efforts addressed in particular the IEEE 802.11 medium access control (MAC) and physical (PHY) layers [5], adding extra functionalities at the MAC layer, such as multiple rate support, rate adaptation support, infrastructure management frames support, IEEE 802.11e support, and including more realistic propagation, interferences, capture and packet loss models at the physical layer. The reader is referred, among others, to [2, 27, 9, 23]. Some of these extensions have now been included in the official ns-2 distribution[1]. At the network layer, new protocols or modified versions of already imple-

*"Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence" appointed by The Research Council of Norway, funded by the Research Council, NTNU, UNINETT and Telenor. http://www.q2s.ntnu.no


*WNS2,* October 24, 2008, Athens, Greece.


[1]At the time of writing, the latest version is ns-2.33.

mented protocols are continuously developed. On the other hand, the design of the network layer itself has not evolved and needs to be extended in order to enable the simulation of the emerging classes of networks introduced above. In particular, support for multiple interfaces, potentially of different types, support for multiple routing and forwarding protocols running on one or more interfaces and support for internetworking between those protocols is required. The objective of this paper is to motivate and present a network layer architecture that supports the aforementioned features and provides a framework for implementing routing and forwarding protocols.

The rest of this paper is organized as follows. Taking the example of wireless mesh networks, Section 2 presents some open research issues at the network layer and we derive a set of required features in order to be able to simulate such networks. In Section 3, we give a brief description of the current design of the network layer and the implementation of routing and forwarding protocols in ns-2 and discuss the extensions needed with regard to the required features listed in Section 2. We then present our proposed architecture in Section 5 and illustrate the functionality provided with a simple simulation in Section 6. Related work, in particular the ns2-MIRACLE framework [7], is reviewed in Section 7, and finally a summary is given in Section 8. In the following, the reader is assumed to be familiar with the ns-2 internals and is referred to [25] for additional details.

## 2. REQUIRED FEATURES

In this section, we focus on one of the applications presented in the introduction, namely wireless mesh networks. We describe briefly this emerging class of networks and outline some of the key challenges and open research issues. The reader may refer to [17] and [11] and the references therein for a more comprehensive description. Finally, we derive a list of features required in order to be able to simulate such networks.

Wireless mesh networks consist of stationary and grid-powered wireless mesh routers, which self-organize to build a wireless backbone and provide global connectivity through a limited number of routers connected to one or more external networks, e.g. fixed networks. See Figure 1 for an illustration. Such networks are expected to provide flexible wireless backhauls with high performance and dependability and able to accommodate high traffic volumes. One of the major difference between mobile ad-hoc networks and wireless mesh networks is the traffic pattern. Most of the traffic is expected to be concentrated toward and from the gateway routers, but at the same time if multiple gateways are available any of them may be used to provide global connectivity.

Key challenges include maximizing the capacity of stationary multi-hop wireless networks, providing support for provisioning of QoS guarantee and QoS differentiation, and exploiting multiple paths for resilience and load balancing. A promising approach to providing high capacity and high redundancy is to equip mesh routers with multiple radio interfaces on non-interfering channels. However, this new degree of freedom also raises some issues. At the network level, open research issues include how to assign channels, how to exploit channel diversity and whether channel assignment should be considered jointly with routing for instance. At the node level, a destination may be reachable via more than one interface, possibly using different technologies, and some criteria are then needed to differentiate routes and to decide which route to use. One approach is to make use of cross-layer information, e.g. some quality measures reported by lower layers [29].

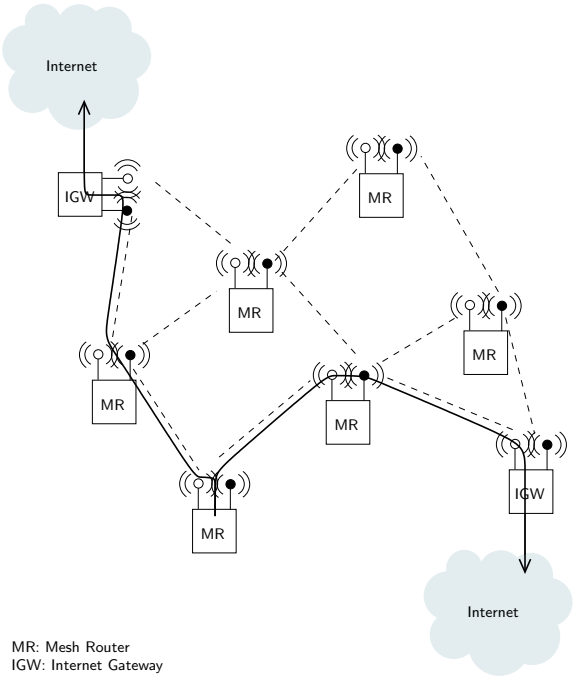


**Figure 1: Example of a multi-hop, multi-interface, multi-channel, multi-destination, multi-routes wireless mesh network**

Studying and understanding such complex multi-* wireless systems and designing and comparing protocols requires advanced simulation tools. In particular, at the network layer, the simulator needs to support multiple interfaces on different channels used by a single routing and forwarding protocol as well as multiple interfaces on different networks, managed by different routing and forwarding protocols. Supporting multiple routes to the same destination and algorithms to decide which route to use is also a required feature.

## 3. CURRENT DESIGN

We now review the current design and implementation of routing and forwarding protocols in ns-2 with emphasis on the support for multiple interfaces, multiple routes and scenarios combining wired and wireless networks. In particular, we stress the fact that, not only depending on the type of routing/forwarding protocol, but also on the type of interface, the implementation details and the node models differ significantly.

### 3.1 Routing and forwarding protocols

If we refer to the ns-2 documentation [25], every routing implementation should consist of three building blocks, namely a routing agent, a route logic and a classifier. Their purpose is respectively to generate and receive routing packets, to use the information gathered by agents to perform the actual route computation, and to forward packets according

to the computed routes. In addition, routing modules should be used to manage these functions and to interface between the node and the classifiers. This is the general framework. In the following, we discuss the actual implementation of routing and forwarding protocols. Before going any further, it is worth mentioning that ns-2 uses the term routing to refer both to the routing functionality itself, i.e. finding routes, but also to the forwarding functionality, i.e. using routes to forward packets.

Ns-2 distinguishes between two types of protocols:

- *Centralized routing protocols* refers to the computation of routes at the simulator level. In other words, the simulator has full knowledge of the network topology and run a user-defined algorithm on this topology to compute routes. Those routes are then pushed instantaneously to all the nodes in the simulation that start using them to forward packets. Such protocol is suited when routes are needed but the routing protocol itself is not the purpose of the study and does not affect the results.
- *Detailed dynamic routing protocols* refers to routing protocols running on each node, sending and receiving routing packets and computing routes in a distributed fashion based on the information gathered. Such protocols are suited when the routing protocol itself or the effect of the protocol dynamics is the purpose of the simulation study.

In addition, *manual routing* refers to the possibility for the user to set routes instead of computing routes.

Since the implementations of protocols for wired and wireless networks differ significantly, we address them separately. For the sake of simplicity, we present only unicast protocols. Table 1 summarizes the description given below.

### 3.1.1 Routing protocols for wired networks

Ns-2 includes centralized and detailed dynamic routing protocols and supports manual routing for wired networks.

Examples of centralized routing protocols are `Static`, the default route computation strategy in ns-2, and `Session`. Both routing protocols use the Dijkstra's all-pairs SPF algorithm [10]. The only difference is that routes are recomputed after a topology change, e.g. a link failure, when `Session` is used, while routes are only computed prior to the start of the simulation when `Static` is chosen. Concretely, the route computation is implemented in C++ in a `RouteLogic` object. No routing packets are simulated, thus no routing agent is attached to each node. To this regard, the fact that the `Static` and `Session` protocols are implemented as routing agents (`Agent/rtproto/Static` and `Agent/rtproto/Session`, respectively) can be misleading. This is only to provide a common way of specifying which routing protocol is used as described below. Those two classes are purely static and no such agent is instantiated.

Examples of detailed dynamic routing protocols in ns-2 are `DV` (Distance Vector) and `LS` (Link State). Concretely, routing packet generation and route computation are implemented in a routing agent (`Agent/rtproto`) in C++ and in OTcl, respectively. More than one such protocol can be running on a single node and hence more than one such agent can be attached to a node. An `rtObject`, written in OTcl only, is also associated to each node. Its purpose is to maintain the forwarding table of the node and to decide which routes to install in the classifiers. Multiple routes to the same destination may be found when more than one routing protocol is running on a node. In this case, routes are assigned preferences and the most preferred route is chosen. If two routes have the same preference value, the route with the lowest cost is chosen. In case of a tie, a random route is installed.

Manual routing is implemented as a routing module which provides the user commands to enter routes. A routing agent is also implemented (`Agent/rtproto/Manual`), but, as for `Static` and `Session`, this is only to provide a common way of defining which routing protocol is used.

To configure the routing protocol to run, the `rtproto` command of the `Simulator` class is used with the name of an `Agent/rtproto` as a parameter. In the case of a centralized routing protocol, only one protocol can be used for all the nodes. When a detailed dynamic routing is used, multiple routing protocols can be used, and the nodes hosting a protocol can be specified as parameters. If no nodes are given, the routing protocol will be running on all nodes, that is a routing agent will be created and attached to every node in the simulator. Routing agents and `rtObject`s are created at the beginning of the simulation, after all nodes have been created. Routing modules are enabled/disabled using the `node-config` command and a module-specific option. Enabled modules are created and attached to the node at its creation.

### 3.1.2 Routing protocols for wireless networks

All the routing protocols for wireless networks included in the official ns-2 distribution are detailed dynamic routing protocols[2]. Examples include AODV (Ad-hoc On-demand Distance Vector routing, [21]) and DSR (Dynamic Source Routing [16]). Their implementation does not follow the general framework described in the introduction to this section and differs significantly from the implementation of detailed dynamic routing protocols for wired networks described above.

All the functionalities - routing packet generation, route computation and packet forwarding - are implemented in a single routing agent object. Although the same term is used to refer to both constructs, routing agents for wireless routing protocols inherit directly from `Agent` and not from `Agent/rtproto` as routing agents used for routing protocols for wired networks. There is no routing module, no classifier and no route logic object. In addition, routing agents for wireless networks are to be attached to `MobileNode` objects. The `MobileNode` is a specialized version of the `Node` built around the routing agent where the routing agent interfaces between the standard node model and a stack of low layers (link layer (LL), and MAC and PHY layers).

The `adhocRouting` option of `node-config` command is used to specify which routing protocol to run on a node. This option takes the name of a routing protocol as a parameter, e.g. AODV. The routing agent is then created at the creation of the node. This option is also what decides whether the created node is a `Node` or a `MobileNode` (or any type of node derived from the `MobileNode`).

For the sake of completeness, it must also be mentioned that extensions to the main code have been developed to

[2]Except the old and outdated implementation of diffusion which uses the general operations director (`God`) as a centralized routing protocol.

| | Baseline ns-2 | | | | Proposed architecture |
|---|---|---|---|---|---|
| Network type | Wired | | | Wireless | Any |
| Protocol type | Centralized | Detailed | Manual | Any | Any |
| Routing packet generation | N/A | `Agent/rtproto` [1 per `Node`] | N/A | `Agent` [1 per `MobileNode`] | `RoutingPacketGenerator` [0+ per `NetworkLayerUnit`] |
| Route computation | `RouteLogic` [1 per `Simulator`] | *`Agent/rtproto`* [1 per `Node`] | | | `RoutingUnit` [0/1 per `NetworkLayerUnit`] |
| Forwarding table | | *`rtObject`* [1 per `Node`] | | | `FIB` [1 per `Node`] |
| Data packet forwarding | `Classifier` [1+ per `RtModule`] | `Classifier` [1+ per `RtModule`] | `Classifier` [1+ per `RtModule`] | | `ForwardingUnit` [1 per `NetworkLayerUnit`] |

*Cursive* typesetting is used for classes/features implemented mostly or fully in OTcl.
1 means exactly one, 0/1 means none or one, 0+ means none or more, and 1+ one or more.

**Table 1: Implementation of routing and forwarding functionalities in ns-2**

enable centralized routing protocol and manual routing for wireless networks, e.g. Adhoc Static Routing (ASR) [24] and No Ad-Hoc routing agent (NOAH) [26], respectively.

## 3.2 Wired-cum-wireless scenarios

Ns-2 provides some support for what is referred to as "wired-cum-wireless" scenarios, that is scenarios where several wireless networks are connected through wired nodes. The idea is to define different wireless domains and one or more base-station nodes in each domain through which all packets to/from destinations outside the domain are sent/ received. In reality, the base station node is nothing but a `MobileNode` connected to one or more wired links. The user specifies which nodes should run the routing protocol for wired networks by turning on the `wiredRouting` option of the `node-config` command before creating those nodes. The implementation relies exclusively on hierarchical addresses and no more than one base station is supported per mobile node or, in other words, the base station to which a packet is transmitted does not depend on its destination. This feature is implemented on a rather ad-hoc and per-protocol manner. Only DSR and DSDV (Highly dynamic Destination-Sequenced Distance-Vector routing, [22]) implementations in the official distribution support this functionality. Some external implementations of AODV also provide this functionality, for instance AODV+ [13] or AODV-UU [18].

# 4. NEEDED EXTENSIONS

In this section we point out needed extensions to the current design with respect to supporting multiple interfaces, potentially of different types, and multiple routing and forwarding protocols at the network layer. Mirroring the description given in Section 3, we first address the models used for wired networks and for wireless networks. We next discuss the interworking of wired and wireless protocols on a node. Finally, we shed some light on more general extensions needed when it comes to supporting multiple interfaces at the network layer.

## 4.1 Wired model

*1) The execution of a routing/forwarding protocol should not depend on neighboring nodes and it should be possible to specify on which interface(s) a protocol is running.* Currently, there is no way of specifying on which interface(s) a detailed dynamic routing protocol should be running. For `DV` and `LS` this is decided at the initialization of the routing agent for each link based on whether the neighboring node is running the same protocol or not. For this, a static list of neighbors built up when creating links and attaching nodes to links is used. This approach cannot be generalized. In the case of a wireless interface, several neighbors may be reached through the same interface and the neighborhood of a node may vary in time. In addition, in a multi-interface multi-channel context, a neighbor may be reachable through more than one interface. Therefore, the execution of a protocol should not depend on neighboring nodes and it should be possible to specify on which interface(s) a routing/forwarding protocol is running.

*2) An entry in the forwarding table needs to contain information about both the next hop, the channel and the output interface used to reach this next hop.* The `rtObject` assumes a mapping between output interface and next hop. This is true in the case of a point-to-point link but cannot be generalized to any type of interface. For instance, a wireless interface does not define the next hop. Again, this is because more than one neighbor may be reached through the same interface and, in a multiple channels context, because a neighbor may be reachable through more than one interface. Thus, a forwarding entry needs to contain information about both the next hop and the output interface used to reach this next hop. In addition, in a multi-channel context, if interfaces are allowed to switch channels dynamically, the mapping between interface and channel cannot be assumed either. Hence, an entry in the forwarding table should also contain the information about the channel to use.

*3) Before deciding what to do with a packet, it should be possible to update per-route and/or per-interface information upon reception of a packet.* In many occasions, per-route and/or per-interface information needs to be updated upon reception of a packet. This is the case for instance when protocols maintain soft-states, e.g. route timeouts, when control information is piggybacked on data packets, or when forwarding decisions depend on the load on each link. With the current design, all packets enter the node

via the node entry point and are forwarded inside the node based on their destination address only. In addition, the information of the incoming interface or link is generally not available. Hence, there is no simple way of updating updating per-route or per-interface information, in particular if several routing protocols are running on the same node and several interfaces of different types are attached to this node.

*4) The design should support several routes to the same destination discovered by one or more routing protocols using different interfaces or a single interface but different next hops.* With the current implementation, multiple routes to the same destination are supported only if they are found by a single routing protocol. Classifiers map each destination address to a link or to the next classifier in the chain. If multiple routes to the same destination are discovered, only the most preferred one is installed. Multiple routes to the same destination may be installed using a `MultiPathForwarder` if the `multiPath` option has been turned on and only if those routes have been discovered by the same routing protocol.

## 4.2 Wireless model

*5) To enable interworking with external networks, it should be possible to run more than one routing protocol on a node.* The `MobileNode` is built up around a single routing agent. This agent is attached to the port demultiplexer on a dedicated routing port and the address classifier use a default target to forward packets internally to the agent. Hence, the current design precludes from attaching more than one routing agent to the same node. Only routing protocols operating over wired links can run simultaneously on the same node.

*6) It should be possible to run a routing protocol over several wireless interfaces on a node.* Currently, there is no native support for routing agents running over more than one wireless interface. The routing agent is attached to a single stack of low layers which types are defined prior to the creation of the node. The objects constituting the stack are instantiated before the routing agent is created and attached to the stack. It is the routing agent that is attached to an interface (to the first created, if several). Supporting multiple interfaces and multiple routing protocols requires the opposite semantics, that is attaching interfaces to a routing protocol.

*7) As for the wired model, it should be possible to update per-route and/or per-interface information upon reception of a packet before deciding what to do with a packet.* The layout of the `MobileNode` prevents from piggybacking routing information to data packets and updating route information upon reception of a data packet at the destination. Indeed, at the destination, data packets are not handed to the routing agent but directly to the end agent. As a result, there is no simple way of updating per-route or per-interface information or reading piggybacked information at the destination upon reception of a data packet with a base `MobileNode`. This is because the address classifier lies before the routing agent. So far, this particular limitation has been solved in an ad-hoc and per protocol manner by designing a node model tailored for a given protocol, e.g. the `SRNode` for DSR.

*8) Routing and forwarding functionalities should be clearly separated to make it easier to study and compare protocols and to enable reusing common functions.* Currently, a routing agent for wireless networks implements both routing and forwarding functionalities. The routing agent is the source and the sink of routing packets and performs route computation and data packet forwarding. There is no clear distinction between routing and forwarding. However, on this particular point, protocol designs may be to blame as much as much as ns-2 [19].

## 4.3 Wired-wireless integration

*9) Using a common node and a common framework for implementing routing and forwarding functionalities regardless of the type of protocol and the type of interfaces would highly facilitate the integration of heterogeneous interfaces and protocols at the network layer.* Currently, routing and forwarding protocols for wired and wireless networks use radically different models. See the summary given in Table 1. Routing and forwarding protocols for wired and wireless networks use different objects, are implemented using different languages and configured using different APIs. In addition, node objects differ depending on whether the node has a wireless interface or not as well as on the routing protocol running making it tedious to compare protocols and reuse common building blocks.

*10) When several routing protocols are running on the same node, a common entity needs to know about the protocols and all the discovered routes to decide what to do with a packet.* Currently, there is no manager for all the routing protocols that operate on a node and no common forwarding table. A `MobileNode` can have a wireless interface and be attached to one or more links. A routing protocol will be running on the wireless interface and another on the wired links. However, they will not share any routing information. Since wireless routing agents are not wrapped in routing modules, they will not know about routes installed by other routing protocols, and reciprocally. The `rtObject` coordinates only routing protocols implemented as `Agent/rtproto` operating on links, and does not know about the wireless routing agent. There is no common forwarding table. If a destination is reachable both using a point-to-point link and a wireless interface, there is no way of specifying which route should be chosen. In such case, the route used depends on whether the address classifier, to which links are attached, lies before or after the routing agent. In the case of the standard `MobileNode`, it lies before and therefore the route using a wired link will always be preferred. Another case where a common entity is missing is when no route to a destination is available. Some entity should have global knowledge about the routing and forwarding protocols running on the node and decide what to do with the packet; whether it should be discarded, forwarded on a default route or to a random neighbor, or buffered and a route discovery procedure triggered if a reactive routing protocol is running on the node.

## 4.4 Multiple interface support

*11) In order to support, multiple interfaces, potentially of different types, on a single node, a generic way of interacting with any kind of network interface is needed.* This is needed for instance for controlling the operation (up/down) of every interface attached to a node. This requires a general interface object and a component that knows about all the interfaces attached to a node. Currently, protocols for wired networks use `Link` objects and classifiers C++ pointers onto the first object (`NsObject`) of the link. Routing agents for the `MobileNode` use a pointer onto a link layer

object (down-target, NsObject). Satellite nodes use NetworkInterface and LinkHead objects. There is no general interface object and no record of all the interfaces attached to a node. In addition, to enable cross-layer interactions when low layers are modelled, there should be a generic way of accessing the layers forming an interface. Similarly, such a feature does not exist in the current implementation. The MobileNode keeps a reference onto each object in OTcl, and for satellite nodes pointers to the different layers are kept in a SatLinkHead object in C++.

*12) It should be possible to uniquely identify every interface attached on a node.* Per-interface identifiers are needed to be able to distinguish between packets coming from the same node but sent on different interfaces and address a particular interface on a node. Currently, addresses are assigned to nodes, not to interfaces. Each NetworkInterface attached to a satellite node is assigned a unique identifier, but this identifier is used to tag packets only. Protocols use node addresses or identifiers, but do not use interface identifiers.

# 5. PROPOSED ARCHITECTURE

In this section, we present an architecture that incorporates the extensions described in Section 4 aims at providing the features described in Section 2. More precisely, our contribution is the definition of network interface and network layer objects to provide a framework for implementing routing and forwarding protocols. It includes support for multiple interfaces, multiple destinations, and multiple routing and/or forwarding protocols operating on the same node each over one or more interfaces, potentially of different types. We describe these objects in more details below. The architecture is summarized and confronted to the baseline ns-2 implementation in Table 1. The relations between the required features, the needed extensions and the objects are illustrated in Figure 2.

Section 6 demonstrates the features provided by this architecture in a simple scenario. Furthermore, this architecture has been successfully used to port the Cross-Entropy Ant System (CEAS) module developed at Q2S, NTNU, Norway. An early version of this module is introduced in [28]. In particular, we have ported the simulator used in [14] which includes stochastic forwarding of data packets.

## 5.1 Network interfaces

As underlined in Section 4.4, one of the missing building blocks in ns-2 when it comes to supporting multiple interfaces, potentially of different types, on the same node is a generic network interface object. Such an object is needed in order to have a generic way of addressing interfaces, keeping track of all the interfaces attached to a node and interacting with these interfaces regardless of their type, e.g. controlling their operation (up/down).

We propose a generic NetworkInterface2[3] object. It can be seen as an aggregate of the LinkHead and NetworkInterface objects. It is both a wrapper and an extra layer. As a layer, it tags both incoming and outgoing packets with the interface identifier. As a wrapper, it has a pointer onto all the components of the network stack (from the link layer and down). More precisely, since ns-2 includes two different models of low layers, we distinguish between interfaces that abstract low layers (NetworkInterface2) and interfaces that do model low layers (NetworkInterface2/FullStack). NetworkInterface2/FullStack includes pointers onto each of the layers constituting the interface, hence enabling some form of cross-layer interactions. Examples of implemented interfaces are NetworkInterface2/PointToPoint and NetworkInterface2/FullStack/Wireless, which wrap the ns-2 point-to-point Link object and wireless stack objects, respectively. There is no a priori restriction on the number of interfaces attached to a node.

Interfaces are configured using the iface-config command, e.g. to define layer types and tracing objects, and created when added to a node using the add-interface command of Node. This command takes the NetworkLayerUnit to which the interface will be attached as a parameter.

[3]As a convention, we add the suffix 2 to a class name when the class is meant to replace the existing class of same name.

## 5.2 Network layer

We propose a two-tier network layer composed of

- a NetworkLayerManager, and
- one or more NetworkLayerUnits.

In addition, each node has a forwarding information base (FIB).

### 5.2.1 Network layer manager

The NetworkLayerManager maintains a list of interfaces and a list of network layer units attached to the node. It receives all data packets generated locally by Agents and all incoming data packets. If a packet is not at destination, its purpose is to decide on a per-packet or per-flow basis by which forwarding protocol, i.e. by which NetworkLayerUnit, the packet should be forwarded. This decision is primarily based on the information stored in the FIB of the node. Additional criteria such as user-defined preferences and/or policies may also be used.

The NetworkLayerManager to be attached to a node is specified using the network-layer-manager option of the node-config command. It is created at the creation of the node.

### 5.2.2 Network layer units

A NetworkLayerUnit is an instance of a routing/forwarding protocol operating on a Node and running over one or more interfaces. It contains a forwarding unit and possibly a routing unit. In both directions, packets pass through a NetworkLayerUnit. This way route information may be updated on a packet transmission/reception. All packets are received at a common point and are then dispatched internally. The forwarding unit handles data packets, the routing unit, if any, routing packets (in reality, the type of packets associated to the NetworkLayerUnit). Details of the NetworkLayerUnit structure are shown in Figure 3. A common queue may be used to enable simulation of the processing time.

A forwarding unit implements a forwarding protocol. In the simplest case, this amounts to handing the packet to the correct interface when a route to the destination exists in the FIB. Other strategies, e.g. stochastic forwarding, may also be implemented. In the case of a reactive routing protocol, if no route to the destination exists in the FIB, the ForwardingUnit may need to buffer data packets and to trigger a route discovery procedure.

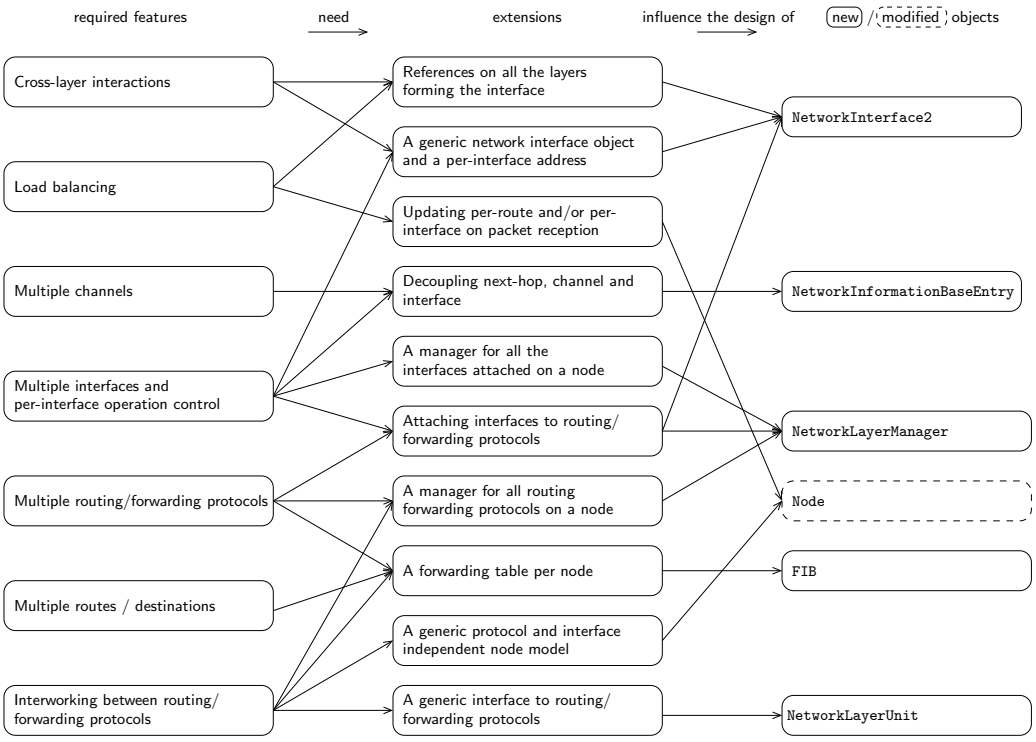


**Figure 2: Illustration of the relations between required features, needed extensions and the objects**

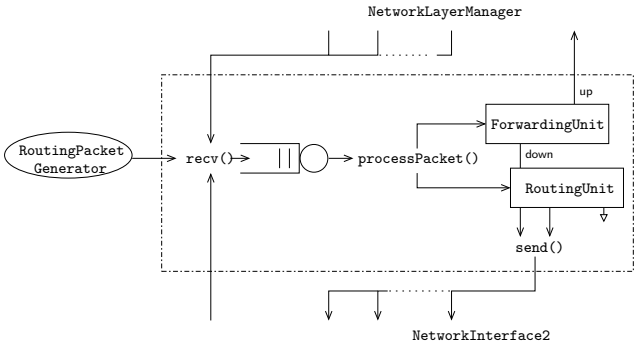


**Figure 3: Detailed structure of a `NetworkLayerUnit`**

A routing unit implements the route logic of a detailed dynamic routing protocol, that is it handles routing packets sent and received by the `NetworkLayerUnit` and performs the route computation. Routing packets are not generated by a `RoutingUnit`, but by one or more `RoutingPacketGenerator`s attached to the `NetworkLayerUnit`. Computed routes are stored internally to the `NetworkLayerUnit` in a routing information base (`RIB`). Which information is contained in the `RIB` highly depends on the protocol. The `RoutingUnit`s are responsible for populating the node `FIB` with stable routes and removing routes when they become stale or unavailable.

### *5.2.3 Forwarding information database*

The `FIB` is used to store routes that can be used to send packets and may contain more than one route to a destination. A route entry (`NetworkInformationBaseEntry`) in the `FIB` contains information about the interface, the next-hop and the channel to use, if relevant. Those routes may have been discovered by any detailed dynamic routing protocol running on the node, computed by a centralized routing protocol or manually added. Support for the two later types of routing is provided by extended `add-route` and `delete-route` procedures of `Node`. Centralized routing protocols such as `Static` or `Session` as well as detailed dynamic routing protocols installing/removing routes using `add-route` and `delete-route` or their C++ counterparts can be used unmodified. On the other hand, routing protocol agents for wireless networks need to be adapted.

## 5.3 Node layout

All the objects described previously are integrated in the standard ns-2 `Node`. A unique object and layout is used regardless of the type of interfaces and the routing/forwarding protocols running instead of using different node models depending on the types of interfaces attached to the node and the routing protocols operating on the node.

For reference, the simplest node layout in ns-2 is recalled in Figure 4. The new layout is shown in Figure 5. Dashed lines represent existing objects and paths. For sake of clarity only one `NetworkLayerUnit` is shown and only one `NetworkInterface2` is attached to the `NetworkLayerUnit`. However, there is no restriction either on the number of `NetworkLayerUnit` on a node or on the number of interfaces attached to such unit. Depending on its type, a `NetworkInterface2` will be attached to a stack of low layers or directly to a `Link`. Figure 5 also shows that how the address classifier, and by extension any legacy chain of classifiers, is integrated in the new structure, thus enabling using existing protocols for wired networks.

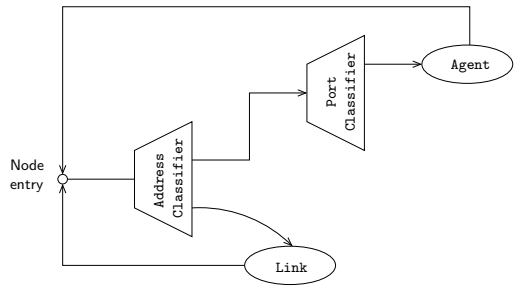


**Figure 4: Schematic representation of a unicast `Node` in ns-2**

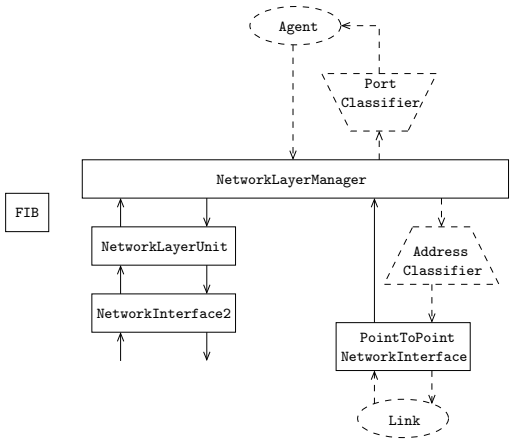


**Figure 5: Proposed `Node` layout**

## 6. SAMPLE SIMULATION SCENARIO

The purpose of this section is to illustrate the functionality provided by the architecture described in Section 5 with a simple example.

We consider the simple multi-* wireless mesh network shown in Figure 6. This network is composed of four nodes; two mesh routers (MR1 and MR2) and two gateways (IGW1 and IGW2) providing connectivity to the Internet. Each mesh router is equipped with two wireless interfaces on two different channels (0 and 1). The gateways only have one interface on channel 0. Each gateway broadcasts periodically connectivity advertisements, which are relayed by its one-hop neighbors. This way every node no farther then

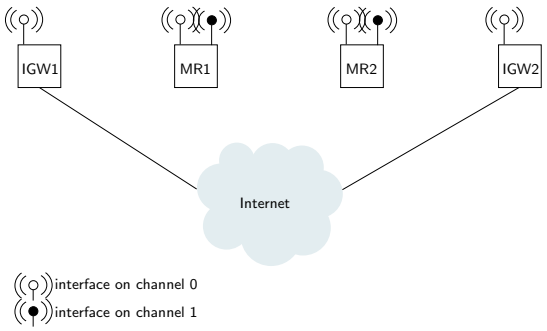


**Figure 6: Example multi-* wireless mesh network**

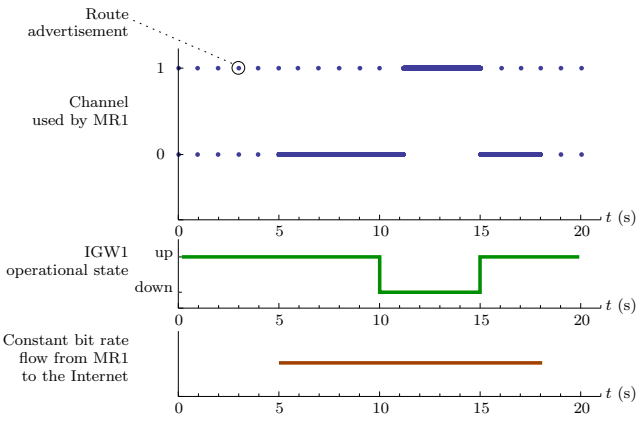


**Figure 7: Sample multi-channel and gateway trace**

two hops from a gateway has at least a route to the Internet. Each route is valid for an inter-advertisement interval. If more than one route to any gateway are available on a node, the route with the minimal number of hops is chosen. For two routes with equal number of hops, the route with the higher channel diversity is preferred.

In this example, IGW1 is down from $t = 10$ s to 15 s. To show how this down-time affects the routes for MR1, we consider a constant bit rate flow from MR1 to the Internet starting at $t = 5$ s and finishing a $t = 18$ s. Figure 7 shows which of the channels are used by MR1 to send packets during the simulation period. IGWs broadcast advertisements every second. When IGW1 is available, MR1 sends data packet directly to IGW1 on channel 0 and relays broadcast advertisement received from IGW1 on both channels. When IGW1 is unavailable, MR1 still has two routes to the Internet and prefers the route through MR2 with the highest channel diversity, i.e. using channel 1 to reach MR2. MR1 continues to use channel 0 after IGW1 is down until the route becomes stale, and it then starts sending packets via MR2 on channel 1.

## 7. RELATED WORK

In [20], we presented the module-based wireless node for ns-2 (MW-Node). We introduced the concept of a unique multi-purpose node with added functionalities, e.g. position and mobility, by means of modules and proposed a better integration of wireless networking in ns-2. Support for multiple wireless interfaces was provided by using a wireless routing module and a chain of classifiers attached to wireless interfaces. However, the implementation was limited to either a single protocol running over several interfaces, or several protocols running over a single interface each and sharing information within the wireless routing module. In addition, different objects were used for implementing either type of protocols. No particular support was provided for sharing information between routing protocols running on different types of interfaces.

To the best of our knowledge, the only recent work concerned with adding more flexibility to ns-2 and supporting multiple interfaces is the ns2-MIRACLE framework [7]. For earlier work on multiple interface support in ns-2, the reader is referred to [20] and the references therein.

The purpose of ns2-MIRACLE[4] is to provide support for a flexible multi-layer node architecture and cross-layer interactions. To this end, authors have developed a generic wrapper for ns-2 objects referred to as `Module`. `Module`s are generalized ns-2 `BiConnector` objects with any number of up- and down-targets. `Module`s are linked together in OTcl to build up `M_Node`s. In addition, `Module`, more precisely `PlugIn` from which `Module` inherits, provides methods for sending and receiving cross-layer messages. Any `Module` is attached to the `NodeCore` thus providing a generic support for cross-layer interactions.

Modules of interest with regard to the work presented in this paper are network layer modules. Ns2-MIRACLE provides a generic `Module/Ip` with address and subnet attributes. Derived objects are `Module/Ip/ Interface` and `Module/Ip/Routing`. `Module/Ip/Interface` does nothing but forwarding up- or downwards packets and discarding packets which are not destined to this node or which cannot be send on this interface. `Module/Ip/Routing` is to be used above one or more `Module/Ip/Interfaces` and provides support for manual routing, i.e. user-defined packet forwarding.

The ns2-MIRACLE framework has been developed within the European project Ambient Networks [1] and has thus been used to simulate typical Always-Best-Connected [12] scenarios. In these scenarios, mobile user devices, equipped with several interfaces using different access technologies, are in the 1-hop range of a base station and should decide which interface to use based for instance on QoS requirements for a given application and quality reported by different layers. Examples of such reports are signal to interference plus noise ratio (SINR) reports from the physical layer, frame loss rate at the MAC layer, delay at the application layer. See for instance the example given in [7]. No routing protocol is needed in such scenarios and to the best of our knowledge no routing protocol has been ported to the ns2-MIRACLE framework. In addition, when several detailed dynamic routing protocols operate on a single node, a module is also needed above all the protocols. In conclusion, in comparison with ns-2, ns2-MIRACLE provides more flexibility and support for cross-layer interactions, but the need for a network layer architecture supporting multiple routing/forwarding protocols and interworking between them is similar.

Another parallel and major recent and ongoing effort is the development of ns-3 [4]. Contrary to our approach, where extensions result from the analysis of what is needed to support new scenarios, network classes and paradigms, the rationale behind the development of ns-3 has been issues and limitations of the ns-2 software itself and its running. The project goals [15] include improved software design, scalability, memory efficiency, support for parallel and distributed simulation, support for emulation and integration of external open-source softwares. This has lead to a node architecture that mirrors real-world IP stacks and will[5] provide support for multiple interfaces (network devices in the ns-3 terminology) and multiple (detailed dynamic) routing protocols. More precisely, we note that the purpose of ns-3 `NetDevice` is similar to that of our `NetworkInterface2` and the purpose of `Ipv4L3Protocol` similar to that of our `NetworkLayerManager`. Therefore, we believe that new network layer objects developed using our framework will be easily portable to ns-3 when it becomes ready for use in network studies.

[4]At the time of writing, the latest version is 1.2.1.

[5]At the time of writing, ns-3 (ns-3.0.12) is still in a pre-alpha state and the core API is not yet frozen.

# 8. CONCLUSION

This paper motivated the design of a network layer architecture extending ns-2 to support the simulation of multi-* networks (multi-technology, multi-homed, multi-hop, multi-interface, multi-channel, multi-destination, multi-route...). Using wireless mesh networks as an example, we derive a list of required features to enable the simulation of such networks. Collating this list with the current design and implementation of ns-2, we then identified the needed extensions. We finally presented our implementation of a network layer architecture which incorporates these extensions and illustrate its functionality with a simple example. In particular, this architecture provides support for multiple interfaces, potentially of different types, and multiple routing and forwarding protocols running on a single node.

More precisely, we defined and implemented a generic network interface object (`NetworkInterface2`) and a two-tier network layer architecture composed of a `NetworkLayerManager` and one or more `NetworkLayerUnit`s. The purpose of the `NetworkLayerManager` is two-fold. First, it serves as a manager for all the interfaces attached to the node and all the `NetworkLayerUnit`s on the node. Second, for each data packet to be forwarded it decides which `NetworkLayerUnit` should forward it. This decision is based on the information contained in the forwarding table of the node and any other criteria such as user-preferences and/or policies. A `NetworkLayerUnit` is a generic container for routing/forwarding protocols attached to any number of network interfaces. It also constitutes a framework for implementing routing/forwarding protocols. These objects are integrated in the standard ns-2 `Node` and an unique layout is used regardless the types of interface attached to the node and the protocols running on this node. Our use so far has indicated that this architecture meets our needs in simulating multi-* networks.

# 9. REFERENCES


[1] The ambient networks project. http://www.ambient-networks.org/.

[2] dei80211mr library. http://www.dei.unipd.it/wdyn/?IDsezione=5091.

[3] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[4] ns-3 project. http://www.nsnam.org/.

[5] IEEE Std. 802.11-2007, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. IEEE Std. 802.11, 2007.

[6] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, Aug. 2002.

[7] N. Baldo, F. Maguolo, M. Miozzo, M. Rossi, and M. Zorzi. ns2-MIRACLE: A modular framework for multi-technology and cross-layer support in network simulator 2. In *Proceedings of the First International Workshop on Network Simulation Tools (NSTools)*, Nantes, France, Oct. 2007.

[8] J. Blum, A. Eskandarian, and L. Hoffman. Challenges of intervehicle ad hoc networks. *IEEE Transactions on*

*Intelligent Transportation Systems*, 5(4):347–351, Dec. 2004.

[9] Q. Chen, F. Schmidt-Eisenlohr, D. Jiang, M. Torrent-Moreno, L. Delgrossi, and H. Hartenstein. Overhaul of IEEE 802.11 modeling and simulation in ns-2. In *Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems (MSWiM)*, Chania, Crete Island, Greece, Oct. 2007.

[10] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerical Mathematics*, 1:269–271, 1959.

[11] V. Gungor, E. Natalizio, P. Pace, and S. Avollone. *Wireless Mesh Networks: Architectures, Protocols, and Applications*, chapter Challenges and Issues in Designing Architectures and Protocols for Wireless Mesh Networks, pages 1–27. Springer-Verlag, 2007.

[12] E. Gustafsson and A. Jonsson. Always best connected. *IEEE Wireless Communications Magazine*, 10(1):49–55, Feb. 2003.

[13] A. Hamidian, U. Körner, and A. Nilsson. Performance of internet access solutions in mobile ad hoc networks. In G. Kotsis and O. Spaniol, editors, *Dagstuhl-Workshop "Mobility and Wireless in Euro-NGI"*, LNCS 3427, pages 189–201, 2005.

[14] P. E. Heegaard and O. J. Wittner. Self-tuned refresh rate in a swarm intelligence path management system. In *Proceedings of the EuroNGI International Workshop on Self-Organizing Systems (IWSOS 2006)*, LNCS, University of Passau, Germany, Sept. 2006. Springer.

[15] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley. ns-3 project goals. In *Proceedings of the 2006 workshop on ns-2: the IP network simulator (WNS2)*, Pisa, Italy, Oct. 2006.

[16] D. Johnson, Y. Hu, and D. Maltz. The Dynamic Source Routing protocol (DSR) for mobile ad hoc networks for IPv4. RFC 4728, 2007.

[17] N. Nandiraju, D. Nandiraju, L. Santhanam, B. He, J. Wang, and D. Agrawal. Wireless mesh networks: Current challenges and future directions of web-in-the-sky. *IEEE Wireless Communications Magazine*, 14(4):79–89, Aug. 2007.

[18] E. Nordström. AODV-UU. http://sourceforge.net/projects/aodvuu/.

[19] E. Nordström, R. Gold, and P. Gunningberg. Mythbusters: Whatever you thought about MANET routing, think again... Technical report, Uppsala University, Sweden, 2005.

[20] L. Paquereau and B. E. Helvik. A module-based wireless node for NS-2. In *Proceedings of the 2006 workshop on ns-2: the IP network simulator (WNS2)*, Pisa, Italy, Oct. 2006.

[21] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) routing. RFC 3561, 2003.

[22] C. E. Perkins and P. Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 24(4):234–244, 1994.

[23] I. Purushothaman and S. Roy. Infrastructure mode support for IEEE 802.11 implementation in NS-2. Technical report, Fundamentals of Networking Lab, University of Washington, USA, 2007.

[24] T. Razafindralambo. ASR - Adhoc Static Routing. http://www2.lifl.fr/ razafind/asr.html.

[25] The VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC. "The *ns* Manual", Kevin Fall and Kannan Varadhan edition, 2005.

[26] J. Widmer. NO Ad-Hoc routing agent (NOAH). http://icapeople.epfl.ch/widmer/uwb/ns-2/noah/.

[27] S. Wiethölter, M. Emmelmann, C. Hoene, and A. Wolisz. TKN EDCA model for ns-2. Technical Report TKN-06-003, Telecommunication Networks Group, Technical University of Berlin, Germany, June 2006.

[28] O. Wittner and B. E. Helvik. Simulating mobile agent based network management using network simulator. Poster in Forth International Symposium on Mobile Agent System (ASA/MA 2000), Sept. 2000.

[29] Q. Zhang and Y.-Q. Zhang. Cross-layer design for qos support in multihop wireless networks. *Proceedings of the IEEE*, 96(1):64–76, Jan. 2008.